به نام خدا

تمرین عملی شماره دو

مبانی بازیابی اطلاعات و جستجو وب

دانیال بیاتی ۹۸۱۶۰۷۳

محمد جواد کفایتی ۹۸۲۳۷۹۳

استاد : دکتر هدی مشایخی

تعدادی مجموعه داده مرجع برای بازیابی قرار دارد که توسط حل تمرین یکی از آنها به گروه شما تعلق میگیرد. در مجموعه داده، تعدادی سند، تعداد پرس و جو و همچنین ارتباط اسناد با پرس و جو ها مشخص شده است. برنامه ای بنویسید که بردار هر سند و هر پرس و جو را در فضای برداری با روش وزن دهی **idf-Tf** به دست آورد. سپس برای هر پرس و جو ۱۰ سند برتر مرتبط با پرس و جو را با روش شباهت کسینوسی محاسبه کنید و با مقایسه با مجموعه مرجع مقادیر صحت، یادآوری و معیار **F ۱** را برای هر پرس و جو و میانگین برای کل مجموعه داده را محاسبه کنید

01

فایل **main.py** برنامه اصلی می باشد که از دو کلاس شباهت کسینوسی و جستوجو تشکیل شده است.

برای حل این تمرین از کتابخانه **whoosh** کمک گرفته شده است که توابع استفاده شده به صورت زیر می باشند:

```python
import math
from collections import Counter
from typing import Dict , List , Any
from typing import Tuple
from whoosh.analysis import StopFilter, StemmingAnalyzer
from whoosh.lang.porter import stem
from whoosh.lang.stopwords import stoplists
import heapq
```

```python
class CosineSimilarity:
    def __init__(self, documents: List[Tuple[str, str]]) -> None:
        """ This function create list of vector for analyze content and scoring

        Args:
            documents (List[Tuple[str, str]]): List of document
        """

        self.doc_freqs : Dict[str, int]  = Counter()
        self.create_doc_frequency(documents)

        self.idf: Dict[str, float]  = {}
        self.create_inverse_term_frequency(documents)

        self.doc_vectors:Dict[int,Tuple[str,List[float]]] = {}
        self.create_document_vectors(documents)

    def create_doc_frequency(self, documents : List[Tuple[str, str]]) -> None:
        """ This function create list of document frequency that contain word and number of
            repeat in all documents

        Args:
            documents (List[Tuple[str, str]]): list of document
        """
        for _, doc in documents:
            self.doc_freqs.update(Counter(doc.split()))

    def create_inverse_term_frequency(self, documents : List[Tuple[str, str]]) -> None:
        """ Create inverse term frequency with this formula
            idf for specified word = log(length of all documents / number of repeats in all docs)

        Args:
            documents (List[Tuple[str, str]]): List of document
        """
        num_docs = len(documents)
        for word, freq in self.doc_freqs.items():
            self.idf[word] = math.log(num_docs / freq)
```

کلاس **CosineSimilarity** :

تابع **create_doc_frequency** به عنوان ورودی لیست داکیومنت های مورد نظر را گرفته و **tf** هر کلمه را محاسبه کرده و آرایه **doc_freq** را بروزرسانی می کند.

تابع **create_inverse_term_frequency** به عنوان ورودی لیستی از داکیومنت ها را دریافت کرده و با پیمایش درون آرایه **docs_freqs** **idf** هرکلمه را حساب کرده و ذخیره می کند.

```python
def create_document_vectors(self, documents: List[Tuple[str, str]]) -> None:
    """ This functions create list of tuples that contain title and
        inverse term frequency each word * term frequency of each word

    Args:
        documents (List[Tuple[str, str]]): List of document
    """
    for i, (title, doc) in enumerate(documents):
        tf = Counter(doc.split())
        vec = [tf[w] * self.idf[w] for w in self.doc_freqs.keys()]
        self.doc_vectors[i] = (title, vec)



def cosine_similarity(self, query :str ="query") -> List[Tuple[str, float]]:
    """ This func create list of the approximate amount query with each document

    Args:
        query (str, optional): A string that the user enters to measure the similarity

    Returns:
        List[Tuple[str, float]]: List of the approximate amount query with each document
    """
    # Removed stop words and finds the root of the rest
    tf: Dict[str, int] = Counter(query.split())
    q_vec:List[float] = [tf[w] * self.idf[w] if w in tf else 0 for w in self.doc_freqs.keys()]
    sims:List[Tuple[str, float]] = []
    for _, (title, d_vec) in self.doc_vectors.items():
        dot_product:float = sum([a*b for a,b in zip(d_vec, q_vec)])
        norm_d:float = math.sqrt(sum([a*a for a in d_vec]))
        norm_q:float = math.sqrt(sum([a*a for a in q_vec]))
        try:
            sim:float = dot_product / (norm_d * norm_q)
        except ZeroDivisionError:
            sim:float = dot_product / ((norm_d * norm_q)+0.0001)
        sims.append((title, sim))

    return sims
```

تابع **create_document_vectors** با توجه به **tf-idf** های حساب شده در دو تابع قبل بردار داکیومنت ها را می سازد تا در مرحله بعدی شباهت کسینوسی محاسبه شود.

تابع **cosine_similarity** پس از محاسبه بردار کوئری مورد نظر با محاسبه ضرب داخلی بردار ها و اندازه هر بردار شباهت داکیومنت ها به کوئری مورد نظر رامحاسبه کرده و درون آرایه **sims** ذخیره می کند تا شباهت همه داکیومنت ها به کوئری را داشته باشیم.

```python
class Searcher:

    def __init__(self ,address_of_documents:str = "./docs/MED.ALL"):
        self.address_of_documents = address_of_documents
        self.read_content_from_docs()
        self.normalize_contents()
        self.cosineSimilarity = CosineSimilarity(self.information)
        self.results: List[Tuple[str, int]] = []

    def preprocessed_text(self, text:str )-> str:
        """This function takes a Text and removes the stop words and finds the root of the rest

        Args:
            text (str): A string of words in a row

        Returns:
            str: A string that removed stop words and finds the root of the rest
        """
        stopwords: Any = frozenset(stoplists["en"])

        analyzer = StemmingAnalyzer(stemfn=stem, stoplist=stopwords) | StopFilter(stoplist=stopwords)
        # Tokenize and analyze the text using the defined analyzer
        tokens:List[str] = [token.text for token in analyzer(text)]

        # Join the tokens back together to form a preprocessed string
        preprocessed_text = " ".join(tokens)

        return preprocessed_text

    def read_content_from_docs(self, address:str = "./docs/MED.ALL"):
        """ This func create list of information of each document that each info includes
            a document number and content

        Args:
            address (str, optional): " Address of file that content documents". Defaults to "./docs/MED.ALL".


        """
        with open(address, 'r') as file:
            documents = file.read()
            list_of_documents = documents.split(".I")[1:]
            # Create list of document number and content
            self.information:Any = [[
                            doc.split(".W")[0].replace("\n","").strip(),
                            doc.split(".W")[1].replace("\n"," ")]
                            for doc in list_of_documents
                        ]
```

کلاس Searcher:

تابع اول همانطور که در کد مشاهده می کنید وظیفه نرمال سازی داکیومنت ها را دارد لذا با استفاده از یه سری عملیات همچون **stemming** و ... داکیومنت های مورد نظر را نرمال می کنیم.

تابع دوم وظیفه خواندن از فایل را برعهده دارد که تمام داکیومنت های موجود در پوشه **docs** و فایل **MED.ALL** به صورت پیششفرض خوانده و با فرمت قابل پیمایش ارائه میدهد

```python
def normalize_contents(self) :
        """ Normalize content of each document with call preprocessed_text func

        Args:
            information (List[List[str]]): List of info that each element is a list that contain a document number and content

        Returns:
            List[List[str]]:  A list that removed stop words and finds the root of the rest
        """
        for index,info in enumerate(self.information):
            document_number , content = info[0],info[1]
            # Update content each info with remove stop words and ...
            self.information[index] = (document_number,self.preprocessed_text(content))

    def search_query(self, query:str)-> List[Any]:
        """ This function is to get similar document to a document

        Args:
            query (str): A text to get similar document

        Returns:
            List[tuple[str,int]]: results of similar document
        """
        query = self.preprocessed_text(query)
        self.results = self.cosineSimilarity.cosine_similarity(query)

    def get_nlargest_similarity_doc(self, n_first:int) ->List[Tuple[str,int]]:
        return heapq.nlargest(n_first, self.results, key=lambda x: x[1])

    def print_results(self, n_first = 10)-> None:
        if self.results != None :
            n_first_similarty_doc = self.get_nlargest_similarity_doc(n_first)
            print('---------------- RESULTS ---------------------')
            print("      SCORE         ***     DOCUMENT NUMBER")
            print('----------------------------------------------')

            for index,result in enumerate(n_first_similarty_doc):
                document_number = result[0]
                score = index + 1
                print(f"      {score}                     {document_number}")
        else :
            print("Please First call search query function")
```

پس از نرمال سازی داکیومنت های موجود شباهت کسینوسی هر داکیومنت را با کوئری مورد نظر حساب کرده و درون آرایه **result** ذخیره می کنیم سپس با توجه به شباهت بیشتر ۱۰ تای بیشتر از نظر شباهت را جدا کرده و در خروجی نمایش می دهیم

```
Please enter your query: the crystalline lens in vertebrates, including humans.
---------------- RESULTS --------------------
      SCORE          ***      DOCUMENT NUMBER
---------------------------------------------
      1                       72
      2                       500
      3                       965
      4                       360
      5                       181
      6                       171
      7                       166
      8                       15
      9                       513
      10                       838
```

```python
s = Searcher()
query = input('Please enter your query: ')
s.search_query(query)
s.print_results(12)
```

تست و بررسی داکیومنت ها :

برای مثال اگر کوئری زیر را داشته باشیم:

query: the crystalline lens in vertebrates, including humans.

خروجی داکیومنت های مورد نظر نشان می دهد که با بررسی فایل **MED.REL** صحت شباهت های به دست آمده به درستی برقرار است.